FLIP ROBO

Rating Prediction

Submitted by:

SARMISHTHA HALDAR

ACKNOWLEDGMENT

I would like to thank my mentor Mr Shubham Yadav for the learning, and guidance provided throughout the course.

INTRODUCTION

- Business Problem Framing

  The problem is related to one such client who has a website where people write different reviews for technical products. Now they are adding a feature to the website Where the reviewer will have to add the stars(rating) as well with the review. The rating is out of 5 starts and it only has 5 options available ie 1star, 2 star 3 star, 4 star and 5 star. Now they want to add a rating for the reviews written in the past but they don't have a rating . So we need to build an application which can predict the rating by seeing the review.

Conceptual Background of the Domain Problem

Ratings help to give us an understanding on how the customers perceive the product Whether they like it or not. What are the pain areas and what is it that they feel about the product? It helps the entrepreneur to have a better understanding of the product and help them in the long run.

- ## Motivation for the Problem Undertaken

The Internet has revolutionized the way we buy products. In the retail e-commerce world of online marketplace, where experiencing products are not feasible. Also, in today's retail marketing world, there are so many new products are emerging every day. Therefore, customers need to rely largely on product reviews to make up their minds for better decision making on purchase. However, searching and comparing text reviews can be frustrating for users. Hence we need better numerical ratings system based on the reviews which will make customers purchase decision with ease.

During their decision-making process, consumers want to find useful reviews as quickly as possible using rating system. Therefore, models able to predict the user rating from the text review are critically important. Getting an overall sense of a textual review could in turn improve consumer experience. Also, it can help businesses to increase sales, and improve the product by understanding customer's needs.

Analytical Problem Framing

Data Sources and their formats

There is a dataset which is formed by collecting the data from various ecommerce websites(Amazon, Flipkart etc). The script used to collect the data has been attached in the zipped file .

```
In [175]: data = pd.read_csv("flipkart1.csv")
```

In [212]:
```
data.head()
```
Out[212]:

	Unnamed: 0	Review	Rating
0	0	sweet thing note buy mostli game make sure cho...	5
1	1	worth cost happi time deliveri wish would glos...	5
2	2	good	5
3	3	go	5
4	4	monitor good deliveri proper 2 month run witho...	4

In [213]:
```
#delete unwanted column
del data['Unnamed: 0']
```
In [217]:
```
data.head()
```
Out[217]:

	Review	Rating
0	sweet thing note buy mostli game make sure cho...	5
1	worth cost happi time deliveri wish would glos...	5
2	good	5
3	go	5
4	monitor good deliveri proper 2 month run witho...	4

- Data Preprocessing Done

  The Data preprocessing involves data cleaning and text preprocessing steps using NLP

  i)Removing the html tags wherever applicable

In [218]:
```
def remove_html(text):
    soup=BeautifulSoup(text,'lxml')
    html_free=soup.get_text()
    return html_free
```

ii)Removing punctuation

```
In [219]: #To remove punctuation
def remove_punctuation(text):
    no_punct="".join([c for c in text if c not in string.punctuation])
    return no_punct
```

```
In [220]: import string
data['Review']=data['Review'].apply(lambda x:remove_punctuation(x))
```

```
In [221]: data['Review'].head()
```

```
Out[221]: 0    sweet thing note buy mostli game make sure cho...
1    worth cost happi time deliveri wish would glos...
2                                                 good
3                                                   go
4    monitor good deliveri proper 2 month run witho...
Name: Review, dtype: object
```

iii)Tokenize

```
In [222]: tokenizer=RegexpTokenizer(r'\w+')
```

```
In [223]: data['Review']=data['Review'].apply(lambda x:tokenizer.tokenize(x.lower()))
```

REMOVE STOPWORDS

```
In [224]: def remove_stopwords(text):
    words=[w for w in text if w not in stopwords.words('english')]
    return words
```

IV)Removing stop words

REMOVE STOPWORDS

```
In [224]: def remove_stopwords(text):
    words=[w for w in text if w not in stopwords.words('english')]
    return words
```


```
In [171]: #nltk.download('stopwords')
```

```
In [225]: data['Review']=data['Review'].apply(lambda x : remove_stopwords(x))
```

V) Stemming and lemmatizing

Stemming and lemmatizing

In [226]:
```
lemmatizer=WordNetLemmatizer()
def word_lemmatizer(text):
    lem_text=[lemmatizer.lemmatize(i) for i in text]
    return lem_text
```

In [227]:
```
#nltk.download('wordnet')
```

In [228]:
```
data['Review']=data['Review'].apply(lambda x : word_lemmatizer(x))
```

Stemmer

In [229]:
```
#Instantiate stemmer

stemmer=PorterStemmer()
```

In [230]:
```
def word_stemmer(text):
    stem_text=" ".join([stemmer.stem(i) for i in text])
    return stem_text
```

In [231]:
```
data['Review']=data['Review'].apply(lambda x : word_stemmer(x))
```

Vi)Ignore the rows which have empty reviews

In [232]:
```
# Sometimes people leave ratings without reviews. We are going to ignore empty reviews.
data = data[data['Review'].isnull()==False]

# Get the ratings column.
ratings = data['Rating']
```

Vii)Rounding off the ratings to 1,2,3,4,5

Rounding off the ratings to make sure they are in numbers 1,2,3,4,5

In [233]:
```
data['Rating']=data['Rating'].replace('-',2)
```

In [234]:
```
data['Rating']=data['Rating'].astype(float)
```

In [235]:
```
def round_school(x):
    i,f=divmod(x,1)
    return int(i +((f >=0.5)if (x >0)else(f> 0.5)))
```

In [236]:
```
data['Rating']=data['Rating'].apply(lambda x : round_school(x))
```

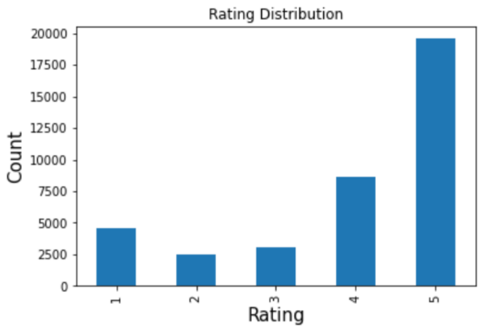
In [237]:
```
data['Rating'].value_counts()
```

Out[237]:
```
5    19558
4     8629
1     4551
3     3050
2     2457
Name: Rating, dtype: int64
```

EDA/ Data Visualization

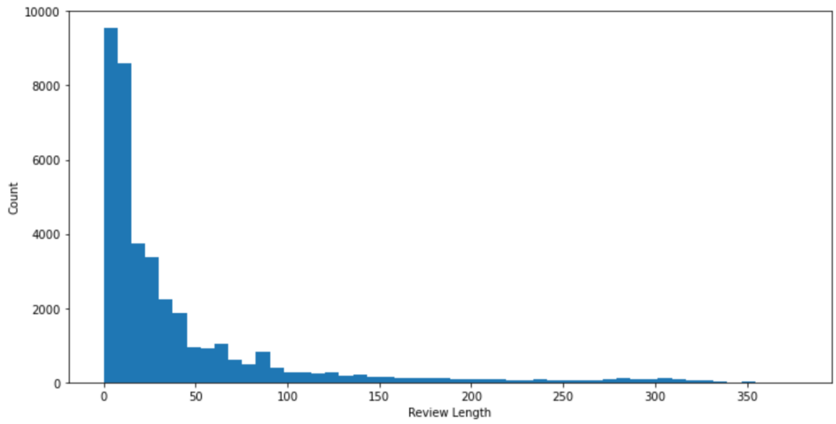
Explore the data

```
In [238]: # Create and print a rating distribution graph.
          plt.figure(figsize = (20, 15))
          rating_distribution_plt = data.groupby(['Rating']).count().plot(kind='bar', legend=None, title="Rating Distribution")
          rating_distribution_plt.set_xlabel("Rating",fontsize=15)
          rating_distribution_plt.set_ylabel("Count",fontsize=15)

Out[238]: Text(0, 0.5, 'Count')

          <Figure size 1440x1080 with 0 Axes>
```



In [239]: # Create and print a Reviews Length distribution graph.

```
In [239]: # Create and print a Reviews Length distribution graph.
          review_length_distribution_plt = pd.DataFrame(data["Review"].str.len())
          review_length_distribution_plt = review_length_distribution_plt[review_length_distribution_plt.Review < 5000]
          review_length_distribution_plt.groupby(["Review"])
          review_length_distribution_plt = review_length_distribution_plt.plot(kind='hist',
                                                                                legend=None,
                                                                                bins=50,
                                                                                figsize=(12, 6))
          review_length_distribution_plt.set_xlabel("Review Length")
          review_length_distribution_plt.set_ylabel("Count")

Out[239]: Text(0, 0.5, 'Count')
```



Rating Prediction - Word

Convert text to Vector

- TF*IDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF*IDF weight of that term

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

TF-IDF VECTORIZER ¶

```
In [298]: from sklearn.feature_extraction.text import TfidfVectorizer
          from sklearn.model_selection import train_test_split
          from sklearn.svm import LinearSVC
          from sklearn.metrics import classification_report

In [299]: tfidf = TfidfVectorizer(max_features=20000, ngram_range=(1,5), analyzer='char')

In [300]: X = tfidf.fit_transform(data['Review'])
          y = df['Rating']

In [301]: y=y.astype('int')

In [302]: X.shape, y.shape

Out[302]: ((38245, 20000), (38245,))
```

Model/s Development and Evaluation

- Testing of Identified Approaches (Algorithms)

Listing down all the algorithms used for the training and testing.

Logistic Regression is the logistic model is used to model the probability of a certain class or event. Logistic regression uses an equation as the representation, very much like linear regression.Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value

MultinomialNB is a naive Bayes classifier. Naive Bayes algorithms are used extensively in text classification problems. They are highly scalable, in fact, MultinomialNB is O(n) (n=training samples size) in training time. MultinomialNB has an alpha hypterparamter that we will set using GridSearchCV. The alpha paramter determines what value we give the when we see a feature that we haven't encountered in the testing data. We can't use 0 because MultinomialNB multiplies these probabilities together and our whole prediction becomes 0. For example, if we find a 5-star review with a single word that we've never seen in a 5-star review before we don't want our prediction to be 0.

SGDClassifier is a very efficient algorithm that uses stochastic gradient descent to build a model. SGDClassifier isn't the model itself; it's a method of building a model. We can specify which model we want to build using the loss parameter. SGDClassifier has many hypterparamters to play with;. The alpha parameter is a constant that multiplies the regularization term, n_iter is the number of passes over the training data.The SGDClassifier guide on sklearn's site8 gives some reasonable ranges to search over

- ## Run and Evaluate selected models

We save 20% of the data for testing. Now we can compare the models by seeing their performance on this data. We can look at their F1-scores, and, to help us visualize their performance, their confusion matrices.

Benchmark model

For the benchmark, we are not going to do any parameter tuning; we are going to use sklearns default parameters. As we will see, logistic regression does fairly well right out of the box.

In [309]:
```
clf_benchmark = LogisticRegression(random_state=22).fit(X_train, y_train)
print( classification_report(y_test, clf_benchmark.predict(X_test), digits=4))
```

```
              precision    recall  f1-score   support

           0     0.8185    1.0000    0.9002       212
           1     0.7053    0.7866    0.7438       928
           2     0.2000    0.0181    0.0332       276
           3     0.4961    0.1881    0.2728       675
           4     0.7121    0.2962    0.4183      1587
           5     0.6958    0.9486    0.8028      3971

    accuracy                         0.6943      7649
   macro avg     0.6046    0.5396    0.5285      7649
weighted avg     0.6682    0.6943    0.6440      7649
```

Support Vector Machine

In [252]:
```
from sklearn.svm import SVR
```

In [253]:
```
svr=SVR()
```

In [254]:
```
svr.fit(X_train, y_train)
```

Out[254]:
```
SVR()
```

In [255]:
```
print(svr.score(X_train,y_train))
```

```
0.7825336118105579
```

In [256]:
```
print(svr.score(X_test,y_test))
```

```
0.6642176536076545
```

In [257]:
```
y_pred=svr.predict(X_test)
```

Other models

Creating an unoptimized MultinomialNB classifier.

In [306]:
```
clf_NB = MultinomialNB()
```

In [307]:
```
clf_NB.fit(X_train, y_train)
```

Out[307]:
```
MultinomialNB()
```

In [308]:
```
print( classification_report(y_test, clf_NB.predict(X_test), digits=4))
```

```
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000       212
           1     0.6017    0.7812    0.6798       928
           2     0.1667    0.0036    0.0071       276
           3     0.3909    0.1274    0.1922       675
           4     0.7057    0.2886    0.4097      1587
           5     0.6551    0.9187    0.7648      3971

    accuracy                         0.6430      7649
   macro avg     0.4200    0.3533    0.3423      7649
weighted avg     0.6000    0.6430    0.5817      7649
```

In [310]:
```
clf_SGD = SGDClassifier(random_state=22)
clf_SGD.fit(X_train, y_train)
print( classification_report(y_test, clf_SGD.predict(X_test), digits=4))
```

```
              precision    recall  f1-score   support

           0     0.7823    1.0000    0.8778       212
           1     0.6807    0.8039    0.7372       928
           2     0.2778    0.0181    0.0340       276
           3     0.4944    0.1970    0.2818       675
           4     0.8505    0.2294    0.3613      1587
           5     0.6865    0.9625    0.8014      3971

    accuracy                         0.6905      7649
   macro avg     0.6287    0.5351    0.5156      7649
weighted avg     0.6908    0.6905    0.6309      7649
```

Hyperparameter tuning

The alpha paramter determines what value we give the when we see a feature that we haven't encountered in the testing data. We can't use 0 because MultinomialNB multiplies these probabilities together and our whole prediction becomes 0. For example, if we find a 5-star review with a single word that we've never seen in a 5-star review before we don't want our prediction to be 0.

In [311]:
```
parameters = { 'alpha': [0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2] }
clf_NB_refined = GridSearchCV(MultinomialNB(), parameters)
clf_NB_refined.fit(X_train, y_train)
```

Out[311]:
```
GridSearchCV(estimator=MultinomialNB(),
             param_grid={'alpha': [0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75,
                                   2]})
```

In [312]:
```
# Print out the parameters GridSearchCV decided on.
clf_NB_refined.best_params_
```

Out[312]:
```
{'alpha': 1.0}
```

compare models

Accuracy and F1 Vscores

In [320]:
```
print("\nAccuracy: ")
print( "LogisticRegression: " + repr(accuracy_score(y_test, clf_benchmark.predict(X_test))) )
print( "MultinomialNB:      " + repr(accuracy_score(y_test, clf_NB_refined.predict(X_test))) )
print( "SGDClassifier:      " + repr(accuracy_score(y_test, clf_SGD.predict(X_test))) )

print("\nclassification reports: ")
print( "LogisticRegression: " )
print( classification_report(y_test, clf_benchmark.predict(X_test), digits=4))
print( "MultinomialNB: " )
print( classification_report(y_test, clf_NB_refined.predict(X_test), digits=4))
print( "SGDClassifier: " )
print( classification_report(y_test, clf_SGD.predict(X_test), digits=4))
```

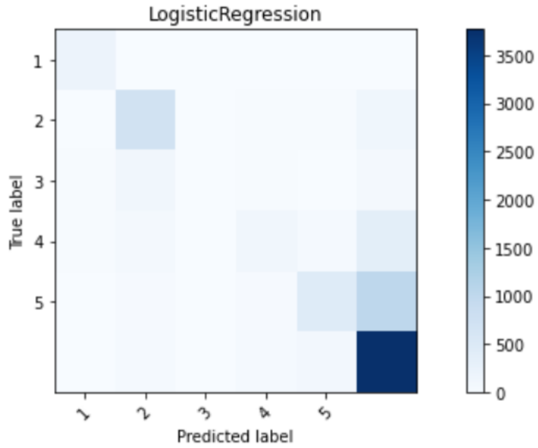
```
Accuracy:
LogisticRegression: 0.6943391292979475
MultinomialNB:      0.6429598640345143
SGDClassifier:      0.6905477840240555

classification reports:
LogisticRegression:
              precision    recall  f1-score   support

           0     0.8185    1.0000    0.9002       212
           1     0.7053    0.7866    0.7438       928
           2     0.2000    0.0181    0.0332       276
           3     0.4961    0.1881    0.2728       675
           4     0.7121    0.2962    0.4183      1587
           5     0.6958    0.9486    0.8028      3971

    accuracy                         0.6943      7649
   macro avg     0.6046    0.5396    0.5285      7649
weighted avg     0.6682    0.6943    0.6440      7649
```

```
MultinomialNB:
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000       212
           1     0.6017    0.7812    0.6798       928
           2     0.1667    0.0036    0.0071       276
           3     0.3909    0.1274    0.1922       675
           4     0.7057    0.2886    0.4097      1587
           5     0.6551    0.9187    0.7648      3971

    accuracy                         0.6430      7649
   macro avg     0.4200    0.3533    0.3423      7649
weighted avg     0.6000    0.6430    0.5817      7649

SGDClassifier:
              precision    recall  f1-score   support

           0     0.7823    1.0000    0.8778       212
           1     0.6807    0.8039    0.7372       928
           2     0.2778    0.0181    0.0340       276
           3     0.4944    0.1970    0.2818       675
           4     0.8505    0.2294    0.3613      1587
           5     0.6865    0.9625    0.8014      3971

    accuracy                         0.6905      7649
   macro avg     0.6287    0.5351    0.5156      7649
weighted avg     0.6908    0.6905    0.6309      7649
```

Confusion Matrices

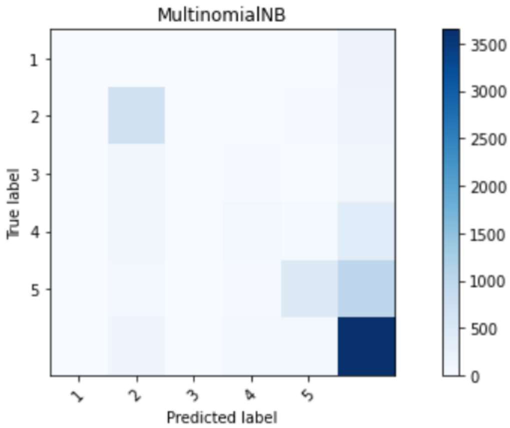
```python
# Create logistic regression confusion matrix.
create_and_print_confusion_matrix(y_test, clf_benchmark.predict(X_test), "LogisticRegression")
```

<Figure size 432x288 with 0 Axes>



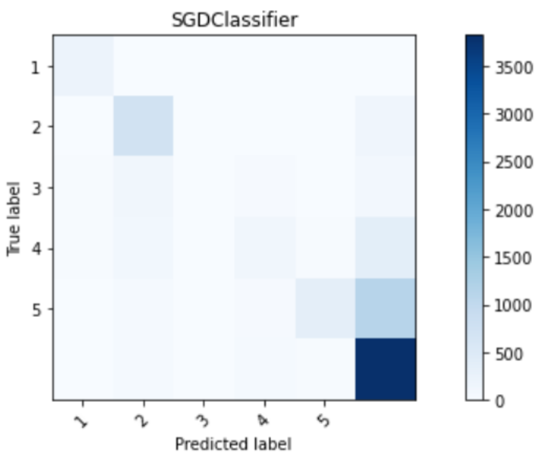
In [332]:
```python
# Create MultinomialNB confusion matrix.
create_and_print_confusion_matrix(y_test, clf_NB_refined.predict(X_test),"MultinomialNB")
```

<Figure size 432x288 with 0 Axes>



In [333]:
```python
# Create SGDClassifier confusion matrix.
create_and_print_confusion_matrix(y_test, clf_SGD.predict(X_test), "SGDClassifier")
```

<Figure size 432x288 with 0 Axes>



- Key Metrics for success in solving problem under consideration

.

The metric we will use to measure the performance of our models is the F1-score. An F1 score is defined as follows:

$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

$Precision = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$

[1] http://www.cs.cornell.edu/home/llee/papers/sentiment.home.html

$Recall = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$

An F1-score is a popular metric to measure the performance of classifiers. An F1-score can be interpreted as a weighted average of precision and recall. [2] The range of an F1-score is between 0 and 1, with the best score being 1.

- Interpretation of the Results, to predict behaviour of independent variable vs target variable

Performance on example reviews

```
In [325]: x = 'this product is really good. thanks a lot for speedy delivery'
          vec = tfidf.transform([x])
          clf_benchmark.predict(vec)

Out[325]: array([5])

In [326]: x = 'this product is very bad. Never buy any product'
          vec = tfidf.transform([x])
          clf_benchmark.predict(vec)

Out[326]: array([1])
```

CONCLUSION

- Conclusion

  After comparing the three models , Logistic regression is supposed to be the best performing .

  Word Cloud

When we look at the top phrases from TF-IDF we should see some similarities with the following generated word cloud:

Reflection
To create this model I took the following steps:
1. The data was downloaded and cleaned.
2. Features were extracted from the review text.
3. The data was split into testing and training sets.
4. A benchmark classifier was created.
5. Several classifiers were trained against the data using automated parameter tuning.
6. The models were tested against the testing data.
7. The models were compared using F1-scoring as a metric.

- # Limitations of this work and Scope for Future Work

All three models have low recall on reviews with star ratings between 2 and 4. Even for a human, it’s difficult to differentiate between these reviews. The difference between a 3-star and 4-star review is very subtle. A good place to look for improvements are ways to improve recall for these mid-star reviews. A solution not explored in this report is a neural network. Many people have found success using neural networks for text classification problems.